Software Fault Tolerance for High-Performance Space Applications

Michael Turmon, Robert Granat, and Daniel S. Katz
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109
tel +1-818-393-5370 fax +1-818-393-5244
{turmon,granat,daniel.katz}@jpl.nasa.gov

Abstract

We describe and test a software approach to overcoming radiation-induced errors in spaceborne applications running on commercial off-the-shelf components. The approach uses checksum methods to validate results returned by a numerical subroutine operating subject to unpredictable errors in data. We can treat subroutines that return results satisfying a necessary condition having a linear form; the checksum tests compliance with this necessary condition. These checksum schemes are called algorithm-based fault tolerance (ABFT). We discuss the theory and practice of setting numerical tolerances to separate errors caused by a fault from those inherent in finite-precision numerical calculations. We test both the general effectiveness of the linear ABFT schemes we propose, and the correct behavior of our parallel implementation of them.

Keywords: ABFT, checksum, parallel, space, software.
Submission category: regular papers.
Word count: approximately 7000 words.
Contact author: Michael Turmon.
This material has been cleared through author affiliations.

1 Introduction

We first outline the general outlook and goals of the spaceborne computing effort motivating this work, and then we describe the detailed contents of this paper.

1.1 Overview

Within NASA's High Performance Computing and Communications (HPCC) Program, the Remote Exploration and Experimentation (REE) project [7] at the Jet Propulsion Laboratory (JPL) will enable a new type of scientific investigation by bringing commercial supercomputing technology into space. Transfer of such powerful computational facilities to space will permit highly autonomous missions with more flexibility and on-board analysis capability, mitigating control latency due to fundamental light-time delays, as well as inevitable bandwidth limitations in the link between spacecraft and ground stations. To do this, REE needs not to develop one computational platform, but to define and demonstrate a process for rapidly transferring commercial high-performance computing technology into ultra-low power, fault-tolerant architectures for space.

Traditionally, spacecraft components have been radiation-hardened to protect against faults caused by the impact of natural galactic cosmic rays and energetic protons. The process of radiation-hardening components generally lowers the clock speed and increases the required power of a component, as well as taking a number of years to complete. This time taken to radiation-harden a component guarantees that it will be long out-of-date by the time that it can be used in space, and has a high cost that must be distributed over a relatively small number of customers. Typically, at any given time, radiation-hardened components have a power:performance ratio that is an order of magnitude lower, and a cost that is several orders of magnitude higher than contemporary commodity off-the-shelf (COTS) components.

The REE project is therefore attempting to use COTS components in space, recognizing that faults will occur in these components, and handling these faults in software. The project consists of three initiatives: applications, computing testbeds, and system software. The purpose of the applications initiative is to demonstrate that the unique high-performance low-power computing capability developed by the project enables new science investigation and discovery. In order to do this, five Science Application Teams (SATs) were chosen to develop scalable science applications, and to port these to REE testbeds running REE system software. The needs of the applications also lead to requirements on the system software, and ensure that the hardware and system software meet the needs of the NASA spaceborne applications community.

Under the testbeds initiative, a system designed to deliver 30 MOPS/watt is currently being built, to be delivered in January 2000. This testbed con-

sists of 40 COTS processors connected by a COTS network fabric. Through future RFPs, the project will obtain additional testbeds that perform faster while using less power. Criteria that are required of the testbeds are: consistency with rapid transfer (18 month or less) of new Earth-based technologies to space, no single point of failure, and graceful degradation in the event of hardware failure.

The system software initiative will provide the services required to let the applications make full use of the hardware while assuring reliable operation in space and providing an easy-to-use development environment. Much like the hardware, the system software is intended to use commercial components as much as possible. The major challenge for the system software is to develop a middleware layer between the operating system and the applications which accepts that both permanent and transient faults will occur and provides for recovery from them.

Five teams comprise the first round of SATs:

- Gamma-ray Large Area Space Telescope (GLAST): This team, led by Prof. Peter Michelson (Stanford) and Prof. Toby Burnett (U. of Washington) will examine detection of gamma rays in a sea of background cosmic rays (about 1 in 10,000 events will be a gamma ray), and reconstruction of the gamma-ray trajectory.

- Mars Rover Science: Dr. R. Steven Saunders (JPL) leads this team, which has two applications. First, texture analysis and image segmentation are used to identify various materials on Mars for further scientific analysis. Second, images obtained from a stereo camera are analyzed for use in autonomous navigation.

- Next Generation Space Telescope (NGST): Led by Dr. John Mather (Goddard Space Flight Center - GSFC), this team also has two applications. The first is to perform multiple fast reads of the charge coupled devices (CCDs) which take the telescope images in order to eliminate or reduce the effect of cosmic rays which hit these CCDs during an exposure. The second is to perform fine optical control by using a wave front sensing algorithm to control a deformable mirror.

- Orbiting Thermal Imaging Spectrometer (OTIS): This team is led by Prof. Alan Gillespie (U. of Washington). They are designing an application to take hyperspectral imaging data and retrieve temperature and emissivity, as well as performing spectral matching and unmixing, then image classification.

- Solar Terrestrial Probe Project (STP): This team, led by Dr. Steven Curtis (GSFC), is examining using fleets of spacecraft for two applications: radio astronomical imaging and plasma moment analysis.

All of these applications take advantage of large amounts of computing, as well as power:performance ratios that are at least an order of magnitude above those available in today's spacecraft. They are attempting to implement and test new approaches to science data processing and autonomy.

The science applications will be used to test, evaluate, and validate candidate architectures and system software. They are generally MPI programs which are not replicated, and therefore, can take full advantage of the computing power of the hardware. (However, REE also intends to support Triple Modular Redundancy (TMR) in software for smaller applications that require high reliability, as opposed to high availability.) As the processors are COTS components, they are not radiation-hardened, and will suffer from faults. (Since memory will be error-detecting and correcting, faults to memory will be largely screened.) Most of the transient faults will be single event upsets (SEUs); their presence requires that the applications be self-checking, or tolerant of errors. SEUs affecting data are particularly troublesome because they typically have fewer obvious consequences than an SEU to code — the latter would be expected to cause an exception.

1.2 Fault Tolerance via Software

It is in this context that we describe and test the mathematical background for using checksum methods to validate results returned by a numerical subroutine operating in an SEU-prone environment. Due to the nature of scientific codes, much of their time is spent in certain common numerical subroutines — as much as 70% in one NGST application, for example. Protecting these subroutines from faults provides one ingredient in an overall software-implemented fault-tolerance scheme. Along the lines laid out above, our general approach has been to wrap existing parallel numerical libraries (ScaLAPACK, FFTW) with fault-detecting middleware. We can treat subroutines that return results satisfying a necessary condition having a linear form; the checksum tests compliance with this necessary condition. These checksum schemes are called algorithm-based fault tolerance (ABFT). Here we discuss the theory and practice of setting numerical tolerances to separate errors caused by a fault from those inherent in finite-precision numerical calculations.

To separate these two classes of errors, we employ well-known bounds on error-propagation within linear algebraic algorithms. These bounds provide

a maximum error that is to be expected due to register effects: any error in excess of this is taken to be the product of a fault. Adapting these bounds to the ABFT setting yields a series of tests having different efficiency and accuracy attributes.

Characteristics of a given scheme are concisely expressed using the standard receiver operating characteristic (ROC) curve. For a given error tolerance, a certain proportion of False Alarms (numerical errors tagged as data faults) and Detections (data faults correctly identified) will be observed. The ROC plots these two proportions parametrically as the tolerance is varied; this describes the performance achievable by a certain detection scheme and provides a basis for choosing one scheme over others. Two series of tests are described here. The first shows the general effectiveness of the linear ABFT schemes we have proposed, and the second verifies the correct behavior of our parallel implementation of them.

1.3 Notation

We close this introduction by introducing some useful notation. Matrices and vectors are written in uppercase and lowercase roman letters; A^{T} is the transpose of the matrix A (conjugate transpose for complex matrices). Any identity matrix is always I; context provides its dimension. A is *orthogonal* if $AA^{\mathsf{T}} = I$. A square matrix is a *permutation* if it can be obtained by re-ordering the rows of I. The size of a vector v is measured by its *p-norm*, a non-negative real number $\|v\|_p$; similarly for matrices A. See [3] (hereafter abbreviated GVL), sections 2.2 and 2.3, for the definitions. The *submultiplicative property* of p-norms implies that $\|AB\|_p \leq \|A\|_p \|B\|_p$ and similarly for vectors.

2 General Considerations

In this paper we are concerned with these operations:

- Product: find the product $AB = P$, given A and B.
- LU decomposition: factor A as $A = PLU$ with P a permutation matrix, L unit lower-triangular, and U upper-triangular.
- Singular value decomposition: factor A as $A = UDV^{\mathsf{T}}$, where D is diagonal and U and V are orthogonal matrices.
- System solution: solve for x in $Ax = b$ when given A and b

- Matrix inverse: given A, find B such that $AB = I$.
- Fourier transform: given x, find y such that $y = Wx$, where W is the matrix of Fourier bases.
- Inverse Fourier transform: given y, find x such that $x = W^{\mathsf{T}}y$.

Although standard numerical packages provide many other routines, the ones above were identified by science application teams as the being of the most interest, partly on the basis of amount of time spent within them.

Each of these operations has been written to emphasize that some linear relation holds among the subroutine inputs and its computed outputs; we call this the *postcondition*. For the product, system solution, inverse, and transforms, this postcondition is necessary and sufficient, and completely characterizes the subroutine's task. For the other two, the postcondition is only a necessary condition and valid results must enjoy other properties as well. In either case, identifying and checking the postcondition provides a powerful sanity check on the proper functioning of the subroutine.

Before proceeding to examine these operations in detail, we mention two points involved in designing ABFT techniques. Suppose for definiteness that we plan to check one $m \times n$ matrix. Any reasonable checksum scheme must depend on the content of each matrix entry, otherwise some entries would not be checked. This implies that simply computing a checksum requires $O(mn)$ operations. Checksum ABFT schemes thus lose their attractiveness for operations taking $O(mn)$ or fewer operations (e.g. trace, sum, and 1-norm) because it is simpler and more directly informative to achieve fault-tolerance by repeating the computation. The second general point is that, although the postconditions above are linearly-checkable equalities, they need not be. For example, the largest eigenvalue of A is bounded by functions of the 1-norm and the ∞-norm, both of which are easily computed but not linear. One could thus evaluate the sanity of a computation by checking postconditions that involve such inequalities. None of the operations we consider requires this level of generality.

The postconditions we consider generically involve comparing two linear maps, which are known in factorized form

$$L_1 L_2 \cdots L_p \stackrel{?}{=} R_1 R_2 \cdots R_q \quad . \tag{1}$$

This check can be done exhaustively via n linearly independent probes for an $n \times n$ system. Of course, exhaustive comparison would typically introduce about as much computation as would be required to recompute the answer

from scratch. On the other hand, a typical fault to data fans out across the matrix outputs, and a single probe would be enough to catch most errors:

$$L_1 L_2 \cdots L_p w \stackrel{?}{=} R_1 R_2 \cdots R_q w \tag{2}$$

for some probe vector w. This is the approach originally recommended by Abraham and his colleagues [4] to implement ABFT in systolic arrays. It has since been extended and refined by several researchers [1, 2, 5, 8].

There are two designer-selectable choices controlling the numerical properties of such an ABFT system: the checksum weights w and the comparison method indicated above by $\stackrel{?}{=}$. When no assumptions may be made about the operands, the first is relatively straightforward: the elements of w should not vary greatly in magnitude so that results figure essentially equally in the check. (In particular, w should be everywhere nonzero.) In what follows, we let w be the vector of all ones; our implementation allows any w to be supplied by the user.

3 Error Propagation

After the checksum vector, the second choice is the comparison method. As stated above, we perform comparisons using the corresponding postcondition for each operation. To develop a test that is roughly independent of the matrices at hand, we use the well-known bounds on error propagation in linear operations. In what follows, we develop a test for each operation of interest. For each operation, we cite a result bounding the numerical error in the computation's output, and then we use this bound to develop a corollary defining a test which is roughly independent of the operands. Throughout, we use $\mathbf{u}$ to represent the numerical precision of the underlying hardware; it is the difference between unity and the next larger floating-point number.

It is important to understand that the error bounds given in the results are *qualitative* and determine the general characteristics of roundoff in our checksum implementations. The estimates we obtain in this section are bounds based on worst-case scenarios, and will typically predict roundoff error larger than practically observed. (See GVL, section 2.4.6, for more on this outlook.) In the ABFT context, using these bounds uncritically would mean setting thresholds too high and missing some fault-induced errors. Their value for us, and it is substantial, is to indicate how roundoff error scales with different inputs. This allows ABFT routines the opportunity to factor out the inputs, yielding performance that is more nearly input-

independent. Of course, some problem-specific tuning will likely improve performance. Our goal is to simplify this tuning process as much as possible.

Result 1 *Let* $\hat{P} = \texttt{mult}(A, B)$ *be computed using a dot-product, outer-product, or gaxpy-based algorithm. The error matrix* $E = \hat{P} - AB$ *satisfies*

$$\|E\|_\infty \leq n\|A\|_\infty\|B\|_\infty \mathbf{u} \tag{3}$$

Proof. See GVL, section 2.4.8. □

Corollary 2 *An input-independent checksum test for* `mult` *is*

$$d = \hat{P}w - ABw \tag{4}$$

$$\|d\|_\infty/(\|A\|_\infty\|B\|_\infty\|w\|_\infty) \overset{>}{<} \tau\mathbf{u} \tag{5}$$

where τ *is an input-independent threshold.*

The test is expressed as a comparison (indicated by the $\overset{>}{<}$ relation) with a threshold; the latter is a scaled version of the floating-point accuracy. If the discrepancy is larger than $\tau\mathbf{u}$, a fault would be declared, otherwise the error is explainable by roundoff.

Proof. The difference $d = Ew$ so, by the submultiplicative property of norms and result 1,

$$\|d\|_\infty \leq \|E\|_\infty\|w\|_\infty \leq n\|A\|_\infty\|B\|_\infty\|w\|_\infty\mathbf{u}$$

and the dependence on A and B is removed by dividing by their norms. The factor of n is unimportant in this calculation, as noted in the remark beginning the section. □

For the remaining operations, we require the notion of a *numerically realistic* matrix. The reliance of numerical analysts on certain proven algorithms is based on the rarity of certain pathological matrices that cause, for example, pivot elements in decomposition algorithms to grow exponentially. Even algorithms regarded as stable and reliable can be made to misbehave when given such unlikely inputs. Because the underlying routines will fail under such pathological conditions, we may neglect them in designing an ABFT system; such a computation is doomed even without faults. Accordingly, the results below must assume that the inputs are numerically realistic to obtain usable error bounds.

Result 3 *Let* $(\hat{P}, \hat{L}, \hat{U}) = \texttt{lu}(A)$ *be computed using a standard* LU *decomposition algorithm with partial pivoting. The backward error matrix* E *defined by* $A + E = \hat{P}\hat{L}\hat{U}$ *satisfies*

$$\|E\|_\infty \leq 8n^3 \rho \|A\|_\infty \mathbf{u} \tag{6}$$

where the growth factor ρ *depends on the size of certain partial results of the calculation, and is bounded by a small constant for numerically realistic matrices.*

Proof. See GVL, section 3.4.6. □

We note in passing that this is close to the best possible bound for the discrepancy, because the error in simply writing down the matrix A must be of order $\|A\|\mathbf{u}$.

Corollary 4 *An input-independent checksum test for* `lu` *as applied to numerically realistic matrices is*

$$d = \hat{P}\hat{L}\hat{U}w - Aw \tag{7}$$

$$\|d\|_\infty / (\|A\|_\infty \|w\|_\infty) \overset{>}{<} \tau\mathbf{u} \tag{8}$$

where τ *is an input-independent threshold.*

Proof. We have $d = Ew$ so, by the submultiplicative property of norms and result 3,

$$\|d\|_\infty \leq \|E\|_\infty \|w\|_\infty \leq 8n^3 \rho \|A\|_\infty \|w\|_\infty \mathbf{u} \quad .$$

As before, the factor of $8n^3$ is unimportant in this calculation. For numerically realistic matrices, the growth factor ρ is bounded by a constant, and the indicated test is recovered by dividing by the norm of A. □

Result 5 *Let* $(\hat{U}, \hat{D}, \hat{V}) = \texttt{svd}(A)$ *be computed using a standard singular value decomposition algorithm. The forward error matrix* E *defined by* $A + E = \hat{U}\hat{D}\hat{V}^\mathsf{T}$ *satisfies*

$$\|E\|_2 \leq \rho \|A\|_2 \mathbf{u} \tag{9}$$

where ρ *is a constant not much larger than one for numerically realistic matrices* A.

Proof. See GVL, section 5.5.8. □

Corollary 6 *An input-independent checksum test for* `svd` *as applied to numerically realistic matrices is*

$$d = \hat{U}\hat{D}\hat{V}^{\mathsf{T}}w - Aw \tag{10}$$

$$\|d\|_2/(\|A\|_2\|w\|_2) \lesssim \tau\mathbf{u} \tag{11}$$

where τ is an input-independent threshold.

Proof. We have $d = Ew$ so, by the submultiplicative property of norms,

$$\|d\|_2 \leq \|E\|_2\|w\|_2 \leq \rho\,\|A\|_2\|w\|_2\mathbf{u}$$

and the dependence on A is removed by dividing by its norm. The constant ρ is negligible for numerically realistic matrices. □

Corollary 7 *An input-independent checksum test for* `svd` *as applied to numerically realistic matrices is*

$$d = \hat{U}\hat{D}\hat{V}^{\mathsf{T}}w - Aw \tag{12}$$

$$\|d\|_\infty/(\|A\|_\infty\|w\|_\infty) \lesssim \tau\mathbf{u} \tag{13}$$

where τ is an input-independent threshold.

Proof. By corollary 6, the check above with the 2-norm in place of the ∞-norm is an input-independent checksum test. But since these two norms are equivalent in that

$$\|w\|_\infty \leq \|w\|_2 \leq \sqrt{n}\|w\|_\infty$$
$$(1/\sqrt{n})\|A\|_\infty \leq \|A\|_2 \leq \sqrt{m}\|A\|_\infty$$

(see GVL sections 2.2.2 and 2.3.2), the two tests are also equivalent up to negligible constants. □

Result 8 *Let $\hat{B} =$* `inv`*(A) be computed using Gaussian elimination with partial pivoting. The backward error matrix E defined by $(A+E)^{-1} = \hat{B}$ satisfies*

$$\|E\|_\infty \leq 8n^3\rho\,\|A\|_\infty\mathbf{u} \tag{14}$$

with ρ as in result 3.

Proof. See GVL, section 3.4.6, which defines the backwards error for the linear system solution $Ax = b$. Since A^{-1} is calculated by solving the multiple right-hand-side problem $AA^{-1} = I$, the bound given there on $\|E\|_\infty$ applies here with the same growth factor ρ. (This growth factor depends only on the pivots in the LU factorization which underlies the inverse computation.) □

Corollary 9 *An input-independent checksum test for* `inv` *as applied to numerically realistic matrices is*

$$d = w - A\hat{B}w \tag{15}$$

$$\|d\|_\infty / (\|A\|_\infty \|A^{-1}\|_\infty \|w\|_\infty) \lessgtr \tau\mathbf{u} \tag{16}$$

where τ is an input-independent threshold.

Proof. See the appendix. □

We remark that this bound on discrepancy, larger than that for `lu`, is the reason matrix inverse is numerically unstable. We close this section with tests for Fourier transform operations. The $n \times n$ forward transform matrix W contains the Fourier basis functions, recall that $W/\sqrt{n}$ is unitary.

Result 10 *Let $\hat{y} = \texttt{fft}(x)$ be computed using a decimation-based fast Fourier transform algorithm; let $y = Wx$ be the infinite-precision Fourier transform. The error vector $e = \hat{y} - y$ satisfies*

$$\|e\|_\infty \leq n \log_2 n \, \|x\|_\infty \mathbf{u} \tag{17}$$

Proof. See the appendix. □

Corollary 11 *An input-independent checksum test for* `fft` *is*

$$d = (\hat{y} - Wx)^{\mathsf{T}} w \tag{18}$$

$$|d| / (\|x\|_\infty \|w\|_\infty) \lessgtr \tau\mathbf{u} \tag{19}$$

where τ is an input-independent threshold.

Proof. This follows from result 10 after neglecting the leading constant. □

Corollary 12 *An input-independent checksum test for* `ifft` *is*

$$d = (\hat{x} - W^{\mathsf{T}} y)^{\mathsf{T}} w \tag{20}$$

$$|d| / (\|y\|_\infty \|w\|_\infty) \lessgtr \tau\mathbf{u} \tag{21}$$

where τ is an input-independent threshold.

Proof. The proof, very similar to corollary 11, is omitted. □

Recommended Checksum Tests

Algorithm	Δ	σ_1	σ_2	σ_3	Note
`mult`	$\hat{P} - AB$	$\|A\|\|B\|$	$\|\hat{P}\|$	$\|\hat{P}w\|$	—
`lu`	$\hat{P}\hat{L}\hat{U} - A$	$\|A\|$	$\|\hat{P}\hat{L}\hat{U}\|$	$\|Aw\|$	σ_1 easier than σ_2
`svd`	$\hat{U}\hat{D}\hat{V} - A$	$\|A\|$	$\|\hat{U}\hat{D}\hat{V}^{\mathsf{T}}\|$	$\|Aw\|$	σ_1 easier than σ_2
`inv`	$I - A\hat{B}$	$\|A\|\|A^{-1}\|$	$\|A\|\|\hat{B}\|$	$\|A\|\|\hat{B}w\|$	$\|A\hat{B}w\|$ useless
`fft`	$(\hat{y} - Wx)^{\mathsf{T}}$	$\|x\|$	—	—	result is a vector
`ifft`	$(\hat{x} - W^T y)^{\mathsf{T}}$	$\|y\|$	—	—	result is a vector

Table 1: Algorithms considered here, and recommended checksum tests.

4 Implementing the Tests

It is straightforward to transform these results into algorithms for error detection via checksums. The principal issue is computing the desired matrix norms efficiently from results needed in the root calculation. For example, in the matrix multiply, instead of computing $\|A\|\|B\|$, it is more efficient to compute $\|\hat{C}\|$ which equals $\|AB\|$ under fault-free conditions. By the submultiplicative property of norms, $\|AB\| \leq \|A\|\|B\|$, so this substitution always underestimates the upper bound on roundoff error, leading to false alarms. On the other hand, we must remember that the norm bounds are only general guides anyway. All that is needed is for $\|AB\|$ to scale as does $\|A\|\|B\|$; the unknown scale factor can be absorbed into τ.

Taking this one step farther, we might compute $\|\hat{C}w\|$ as a substitute for $\|A\|\|B\|\|w\|$. Here we run an even greater risk of underestimating the bound, especially if w is nearly orthogonal to the product, so it is wise to use instead $\lambda\|w\| + \|\hat{C}w\|$ for some problem-dependent λ. Extending this reasoning to the other operations yields the comparisons in table 1. The error criterion used there always proceeds from the number $\delta = \|\Delta w\|$ for the indicated difference matrix Δ; this matrix is of course never explicitly computed. In addition to the obvious

$$T0: \qquad \delta/\|w\| \gtrless \tau\mathbf{u} \qquad \text{(trivial test)} \tag{22}$$

we provide three other comparison tests

$$T1: \quad \delta/(\sigma_1\|w\|) \gtrless \tau\mathbf{u} \quad \text{(ideal test)} \tag{23}$$

$$T2: \quad \delta/(\sigma_2\|w\|) \gtrless \tau\mathbf{u} \quad \text{(approximate matrix test)} \tag{24}$$

$$T3: \quad \delta/(\lambda\|w\| + \sigma_3) \gtrless \tau\mathbf{u} \quad \text{(approximate vector test)} \tag{25}$$

The *ideal test* is the one recommended by the theoretical error bounds, and is based on the supplied input arguments, but may not be computable. In contrast, both approximate tests are based on computed quantities, and may also be suggested by the reasoning above. The *matrix test* involves a matrix norm while the *vector test* involves a vector norm and is therefore more subject to false alarms. (Several variants of the matrix tests are available for these operations.) We note that the obvious vector test for `inv` uses $A\hat{B}w$, but since $\hat{B} = \texttt{inv}(A)$, this test becomes essentially equivalent to the trivial test. We therefore suggest using the vector/matrix test shown in the table. Finally, the ideal tests T1 for the Fourier transforms need only the norm of the input, which is readily calculated. Consequently, other test versions are not stated for these operations.

Clearly the choice of which test to use is based on the interplay of computation time and fault-detection performance for a given population of input matrices. Because of the shortcomings of numerical analysis, we cannot predict definitively that one test will outperform another. The experimental results reported in the next section are one indicator of real performance, and may motivate more detailed analysis of test behavior. Performance in application codes will be another criterion for choosing tests.

5 Results: Simulated Fault Conditions

In this section we show results of Matlab simulations of the proposed checksum tests. These simulations are intended to verify the essential effectiveness of the checksum technique for ABFT, as well as to sketch the relative behaviors of the tests described above. Due to the special nature of the population of test matrices, and the shortcomings of the fault insertion scheme, these results *must not* be taken as anything but an estimate of relative performance, and a rough estimate of ultimate absolute performance.

We briefly describe the simulation setup. In essence a population of random matrices is used as input to a given computation; faults are injected in half these computations, and a checksum test is used to attempt to identify

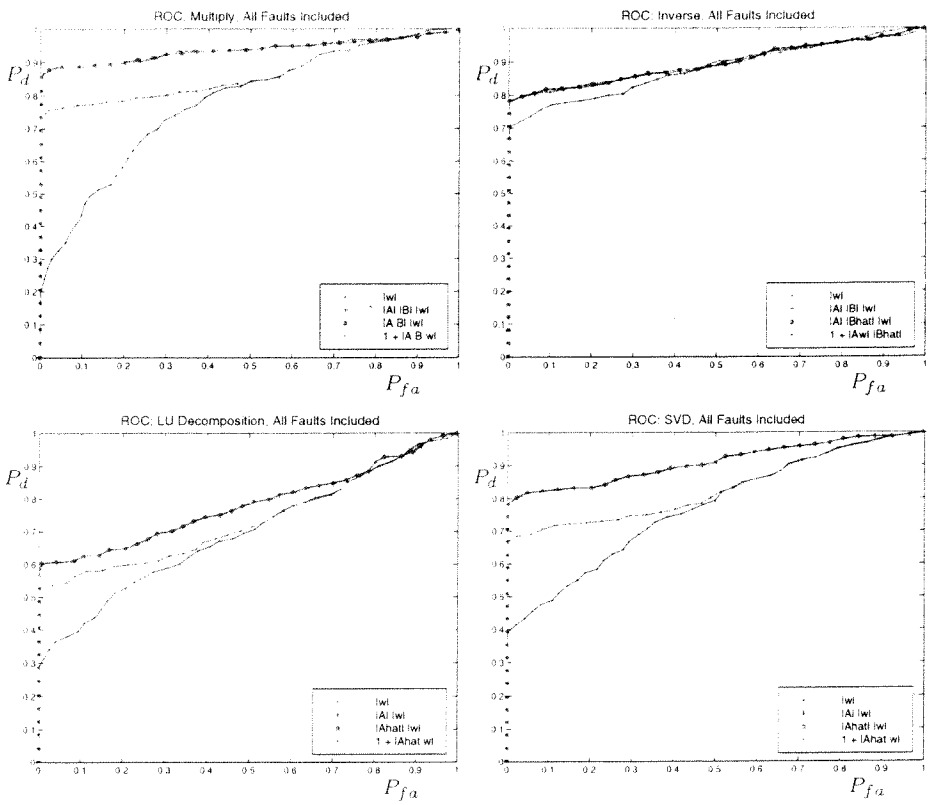

Figure 1: ROC for random matrices of bounded condition number, including all faults.

the affected computations. Random test matrices A of a given condition number κ are generated by the rule

$$A = 10^{\alpha} U D_{\kappa} V^{\mathsf{T}} \quad . \tag{26}$$

The random matrices U and V are the orthogonal factors in the QR factorization of two square matrices with normally distributed entries. The diagonal matrix D_{κ} is filled in by choosing random singular values, such that the largest singular value is unity and the smallest is $1/\kappa$. These matrices all have 2-norm equal to unity; the overall scale is set by α which is chosen uniformly at random between -8 and +8. A total of 800 64×64 matrices (forty applications of the rule (26) for each κ in $\{2^1, ..., 2^{20}\}$) is processed.

Average-case Matrices, Significant Faults

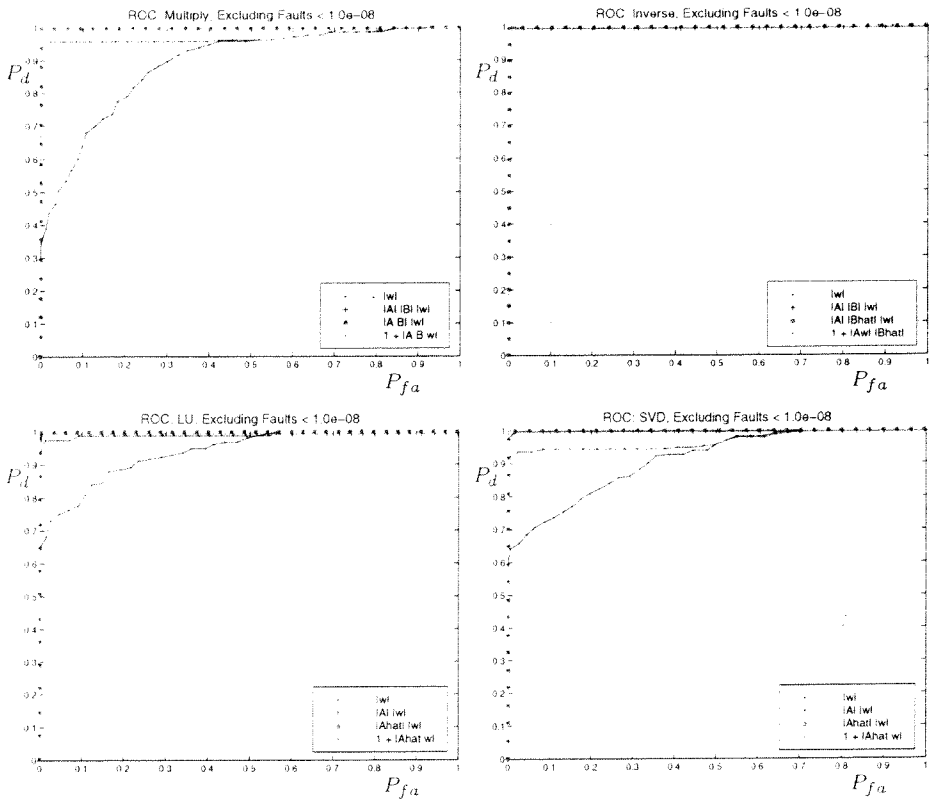

Figure 2: ROC for random matrices of bounded condition number, excluding faults of relative size less than 10^{-8}.

Faults are injected in half of these runs (400 of 800) by first choosing a matrix to affect, and then flipping exactly one bit of its 64-bit representation. For example, if a call to `mult` is to suffer a simulated fault, first A or B is selected at random, and then one bit of the chosen matrix is toggled. If `lu` is to suffer a fault, one of A, L, or U is selected and the fault is injected. If A was selected, one can expect the computed $\hat{L}$ and $\hat{U}$ to have many incorrect elements; if L was selected, only one element of the LU decomposition would be in error. This scheme is intended to simulate errors occurring at various times within the computation.

Next, each of the four tests described above is used to identify faults; for a fixed τ this implies observing a certain false alarm rate and fault-detection

P^* Across Experiments

	Average-Case		Worst-Case	
	All	Sig.	All	Sig.
`mult`	0.86	1.00	0.63	0.92
`inv`	0.78	1.00	0.32	0.50
`lu`	0.60	1.00	0.43	0.90
`svd`	0.78	0.97	0.60	0.87
Mean	0.76	0.99	0.50	0.80

Table 2: Fault detection probability when no false alarms are permitted. Worst-case results are taken from the Matlab gallery matrices. Results when all faults must be detected, and when only significant faults must be detected, are shown.

rate. The pair (P_{fa}, P_d) may be plotted parametrically versus τ to obtain an ROC curve which illustrates the performance achievable by a given test. See figure 1. In these figures, $T0$ and $T3$ are the solid blue lines with dots, with $T0$ in dark blue lying below $T3$. $T2$ is shown in red asterisks, and $T1$, the optimal test, in green crosses.

Of course, some missed fault detections are worse than others since many faults occur in the low-order bits of the mantissa and cause very minor changes in the matrix. Accordingly, a second set of ROCs is shown in figure 2. In this set, faults causing a minute perturbation (less than one part in 10^{-8}, about the accuracy of single-precision floating point) are screened from the results entirely. This curve is more realistic for our applications.

We may make some general observations about the results. Clearly $T0$, the un-normalized test, fares poorly in all experiments. This illustrates the value of the results on error propagation that form the basis for the normalized tests. Generally speaking,

$$T0 \ll T3 < T2 \approx T1 \quad . \tag{27}$$

This confirms theory, in which $T1$ is the ideal test and the others approximate it. In particular, $T1$ and $T2$ are quite similar because generally only an enormous fault can change the norm of a matrix — these cases are easy to detect.

Further, we note that the most relevant part of the ROC curve is when $P_{fa} \approx 0$: we may in fact be interested in the value P^*, defined to be P_d when $P_{fa} = 0$. P^* is the detection rate when no false alarms are permitted; it is summarized for these experiments in table 2. The first two columns of

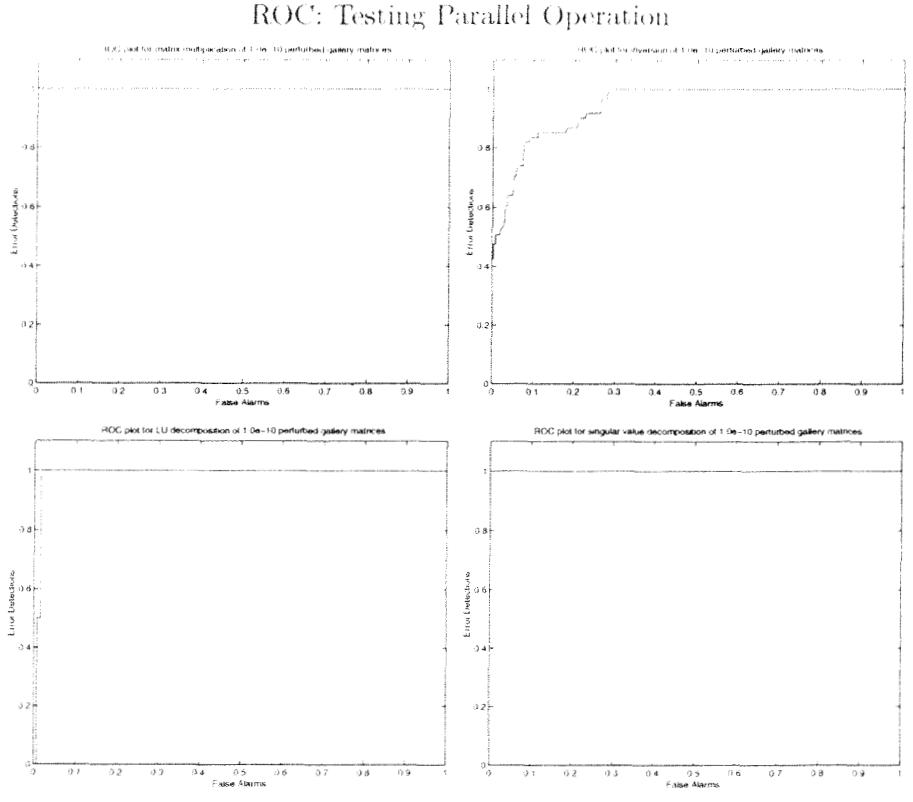

Figure 3: Parallel implementation checked by Matlab computation.

this table come from the data in figures 1 and 2; the other columns are from a "worst-case" matrix population taken from the Matlab gallery matrices. Under the average-case test conditions, about 99% of faults could be detected with no false alarms; this level of performance would seem adequate for REE purposes. In worst-case — and no science application should be in this regime — effectiveness drops to about 80%. This shows that fault detection will be more effective for numerically well-posed applications.

6 Results: Fault-free parallel operation

In our parallel implementation of the checksum procedures we use the ScaLAPACK routines `PDGEMM` for `mult`, `PDGETRF` for `lu`, `PDGETRI` for `inv`, and `PDGESVD` for `svd`. For `mult`, we use the checksum test $T2$ for reasons of com-

putational cost, as the test requires only the calculation of the norm of the resultant matrix product. For `lu` and `svd`, we employ the ideal checksum test $T1$, as in these cases the norm of the matrix can be calculated before the factorization is performed in place. Our choice of checksum test for `inv` is complicated by the fact that `PDGETRI` requires that the input matrix already be in its LU-factorized form. We therefore employ a modified version of the checksum test $T3$, in which $\sigma_3 = \|\hat{B}\|\|Aw\|$: $\hat{B}$ is readily available as the result of the computation, and Aw can be obtained by multiplying w successively by $\hat{U}$, $\hat{L}$, and $\hat{P}$.

We note that in our implementation we consider the possibility that induced faults could affect the calculated norms, thereby compromising the validity of the checksum test. In order to prevent erroneously large norms from eliminating errors from detection, the routines compare the norms against the system dependent maximum double precision floating point value; detection of a norm that exceeds that value results in an error being declared.

In order to address numerical issues concerning our implementation of the checksum procedures, we briefly examine some characteristics of the implementation. Shown in figure 3 are certain ROC curves for the four operations we have considered. In contrast to the results just reviewed, these curves were generated by checking ScaLAPACK computations with Matlab.[1] In this test we use randomly perturbed matrices from the Matlab gallery selection. These matrices are generally ill-conditioned or poorly scaled, but serve as a demanding test set to check our routines against a known standard. In this case, for simplicity, the overall scale parameter $\alpha = 0$ and a fixed perturbation scale $\epsilon = 10^{-10}$ was used.

This time, identical matrix operands are given to Matlab and to our ScaLAPACK implementation. Faults are not injected by modifying operands because our objective is to verify the correct numerical operation of our subroutines. Each system computes the full result matrix; these are combined with the ScaLAPACK checksum comparison to form an ROC as follows. We declare that an error has occurred when the two full results differ by more than a fixed tolerance (10^{-10} in these experiments). An error is declared to have been *detected* or not according to whether a checksum discrepancy was found by the ScaLAPACK implementation. (The Matlab implementation does not compute a checksum; it is used only to find the full result matrix.) With these definitions, a false alarm, for example, means that ScaLAPACK found a checksum discrepancy, but no significant discrepancy was present in

[1] Matlab uses LINPACK `ZGEDI`/`ZGEFA` for `inv` and `lu`, and `ZSVDC` for `svd`. For `mult`, Matlab uses a straightforward inner product implementation with nested loops. [6]

the result of the computation. The ROC thus serves as a check, via Matlab, on the numerical characteristics of our ScaLAPACK implementation. In doing these tests, the comparison rule $T0$ was used; this has smaller consequences than in the previous section because most of the perturbed gallery matrices have roughly unit norm.

These curves were generated by sweeping the threshold τ used in the ScaLAPACK $T0$ test from 0 to ∞. It is clear from the curves that there is excellent agreement between the ScaLAPACK and Matlab versions of `mult`, `lu`, and `svd`. Indeed, when the matrix is badly scaled, ill-conditioned, or numerically unrealistic — causing ScaLAPACK and Matlab to differ according to the full answer — ScaLAPACK finds the error in the checksum calculation also. In essence, the message is: if the computation did not succeed, the checksum test discovers it. Because of the additional instability of the inverse algorithm, its results are less definite. One explanation is that the checksum test is missing some errors that occur in the computed inverse; this needs further investigation.

7 Conclusions

Theoretical results bounding the expected roundoff error in a given computation provide several types of input-independent threshold tests for checksum differences. The observed behavior of these tests is in good general agreement with theory, and readily computable tests are easy to define. All the linear algebra operations considered here (`mult`, `lu`, `inv`, and `svd`) admit tests that are effective in detecting faults at the 99% level on typical matrix inputs. Tests of the numerical characteristics of our parallel implementation of the fault detection schemes indicate excellent agreement with another numerical package for most operations, except in cases when the matrix is badly scaled, ill-conditioned, or numerically unrealistic. In those cases, the schemes detect an error in the checksum calculation.

Test programs calling our parallel implementations have been installed on the REE project testbed, where they can be tested under simulated fault conditions. The fault injector is designed so that in can simulate radiation-induced SEUs affecting memory, registers, code, and the stack. Of the operations described here, `mult` and `fft` have both been tested not only under the protection of ABFT schemes, but also within a software framework such that programs which have crashed or hung are restarted automatically. This works in conjunction with the ABFT error detection: if an error is detected, and the computation does not yield correct results after a set

number of retries, the error handling scheme aborts the program so that it can be automatically restarted.

The ABFT `fft` routines have also been integrated into the image texture analysis and segmentation application which is part of the Mars Rover Science project. This application is being tested with simulated fault injections on the REE project testbed under the software framework described above, both with and without the ABFT routines. While conclusive results are not yet available, preliminary testing indicates that the checksum scheme effectively protects the Fourier transform operations within the application from SEUs.

We expect that continued integration of ABFT routines with the various science applications will lead to these applications being resistant to SEUs throughout large portions of the computation. This is of course just one of the protections that will be needed to use COTS computers in space, but it is an essential one.

Acknowledgment

This work was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

References

[1] D. L. Boley, R. P. Brent, G. H. Golub, and F. T. Luk. Algorithmic fault tolerance using the Lanczos method. *SIAM J. Matrix Anal. Appl.*, 13(1):312–332, 1992.

[2] M. P. Connolly and P. Fitzpatrick. Fault-tolerent QRD recursive least squares. *IEE Proc. Comput. Digit. Tech.*, 143(2):137–144, 1996. IEE, not IEEE.

[3] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins Univ., Baltimore, second edition, 1989.

[4] K.-H. Huang and J. A. Abraham. Algorithm-based fault tolerance for matrix operations. *IEEE Transactions on Computing*, 33(6):518–528, 1984.

[5] F. T. Luk and H. Park. An analysis of algorithm-based fault tolerance techniques. *Journal of Parallel and Distributed Computing*, 5:172–184, 1988.

[6] MathWorks, 14 April 1999. Personal communication via e-mail from J. Regensburger.

[7] REE project, March 1999. "Project Plan: Remote Exploration and Experimentation (REE) Project," available at `www-ree.jpl.nasa.gov`.

[8] Wang and Jah. Algorithm-based fault tolerance for FFT networks. *IEEE Transactions on Computing*, 43(7):849–854, 1994.

Appendix

In this section we present some of the proofs that were passed over in the text.

Proof of Corollary 9. Note that $d = \Delta w$ where $\Delta = I - A(A+E)^{-1}$. Some algebra is necessary to extract the error E from Δ. Using the Sherman-Morrison formula (GVL section 2.1.3) to rewrite the inverse of $A + E$ we obtain

$$\begin{aligned} \Delta &= I - A[A^{-1} - A^{-1}(I + EA^{-1})^{-1}EA^{-1}] \\ &= (I + EA^{-1})^{-1}EA^{-1} \end{aligned} \tag{28}$$

For numerically realistic matrices, A dominates E and the first factor is negligible. Heuristically, this is because $E \ll A$ implies $EA^{-1} \ll AA^{-1} = I$, collapsing that factor to I. More formally, inverting a numerically realistic matrix produces an error matrix E such that for any vector v, $\|Ev\| \ll \|Av\|$ otherwise the backward error E would be comparable to A. Since v is arbitrary and A is invertible, we may let $v = A^{-1}u$, obtaining that $\|EA^{-1}u\| \ll \|u\| = \|Iu\|$, showing that the operator EA^{-1} is dominated by I. Therefore we may neglect the first factor and the norm of the error is bounded by

$$\begin{aligned} \|d\|_\infty &= \|\Delta w\|_\infty \\ &\leq \|E\|_\infty \|A^{-1}\|_\infty \|w\|_\infty \\ &\leq 8n^3 \rho \|A\|_\infty \|A^{-1}\|_\infty \|w\|_\infty \mathbf{u} \end{aligned} \tag{29}$$

using the submultiplicative property of norms. As before, the factor of $8n^3$ is unimportant in this calculation. Invoking the assumption that A is a numerically realistic matrix allows us to neglect the growth factor ρ, yielding the indicated test. □

Proof of Result 10. Decimation algorithms are based on compact factorizations of the $n \times n$ unitary transform matrix W:

$$y = W_N W_{N-1} \cdots W_1 \, x$$

where $N = \log_2 n$, and each W_k performs one bank of $n/2$ "butterfly" operations. The infinite-precision computation may therefore be written as a recurrence

$$\begin{aligned} z_0 &= x \\ z_{k+1} &= W_{k+1} \, z_k \quad (k \geq 0) \end{aligned} \tag{30}$$

where $y = z_N$. The finite-precision computation finds, in turn,

$$\begin{aligned} \hat{z}_0 &= x \\ \hat{z}_{k+1} &= \mathtt{mult}(W_{k+1}, \hat{z}_k) \quad (k \geq 0) \end{aligned} \tag{31}$$

and $\hat{y} = \hat{z}_N$. The proof proceeds by developing a recurrence for the size (always expressed in ∞-norm) of the error vector

$$\begin{aligned} e_{k+1} &= z_{k+1} - \hat{z}_{k+1} \\ &= W_{k+1} \, z_k - \mathtt{mult}(W_{k+1}, \hat{z}_k) \\ &= W_{k+1} \, z_k - (W_{k+1} \, \hat{z}_k + \tilde{e}_k) \\ &= W_{k+1} \, e_k - \tilde{e}_k \end{aligned} \tag{32}$$

where by Result 1, and the observation that exactly two entries of each row of W_k are nonzero, $\tilde{e}_k$ satisfies

$$\begin{aligned} \|\tilde{e}_k\| &\leq 2 \, \|W_{k+1}\| \, \|\hat{z}_k\| \mathbf{u} \\ &= 2 \, \|z_k - e_k\| \mathbf{u} \\ &\leq 2(\|z_k\| + \|e_k\|) \mathbf{u} \quad . \end{aligned} \tag{33}$$

Combining with (32) yields the bound

$$\begin{aligned} \|e_{k+1}\| &\leq \|W_{k+1}\| \|e_k\| + 2(\|z_k\| + \|e_k\|) \mathbf{u} \\ &= (1 + 2\mathbf{u}) \|e_k\| + 2 \|z_k\| \mathbf{u} \quad . \end{aligned} \tag{34}$$

Since $\|W_k\| = 2$, $\|z_k\| \leq 2^k \|x\|$, and we obtain the recurrent upper bound

$$\begin{aligned} \|e_0\| &= 0 \\ \|e_{k+1}\| &\leq (1 + 2\mathbf{u}) \|e_k\| + 2^{k+1} \|x\| \mathbf{u} \quad (k \geq 0) \end{aligned} \tag{35}$$

For any reasonable floating-point system, $1 + 2\mathbf{u} \leq 2$. Using this, it is easy to see $\|e_k\| \leq k \, 2^k \, \|x\| \mathbf{u}$, establishing the claim $\|e_N\| \leq n \log n \, \|x\| \mathbf{u}$. $\square$